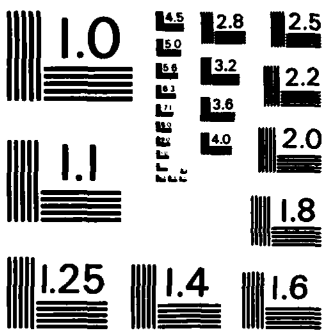
AD-A159 119 A MODELESS CONVEX HULL ALGORITHM FOR SIMPLE POLYGONS (U) CARNEGIE-MELLON UNIV PITTSBURGH PA ROBOTICS INST M A PESHKIN ET AL. MAY 85 CMU-RI-TR-85-8 1/1

UNCLASSIFIED F/G 12/1 NL

END

FILMED

DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD-A159 119

DTIC
ELECTE
SEP 16 1985
S D

Carnegie-Mellon University

The Robotics Institute

THE ROBOTICS INSTITUTE

Technical Report

DTIC FILE COPY

(2)

A Modeless Convex Hull Algorithm for Simple Polygons

M. A. Peshkin and A. C. Sanderson

CMU-RI-TR-85-8

The Robotics Institute
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

DTIC
ELECTE
SEP 1 6 1985
S A

May 1984

Copyright © 1985 Carnegie-Mellon University

This work was supported by a grant from the Xerox Corporation, and by the Robotics Institute, Carnegie-Mellon University.

This document has been approved for public release and sale; its distribution is unlimited.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER CMU-RI-TR-85-8	2. GOVT ACCESSION NO. A159119	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Modeless Convex Hull Algorithm for Simple Polygons		5. TYPE OF REPORT & PERIOD COVERED Interim
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) M. A. Peshkin and A. C. Sanderson		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Carnegie-Mellon University The Robotics Institute Pittsburgh, PA. 15213		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE May 1985
		13. NUMBER OF PAGES 7
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Approved for public release; distribution unlimited

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

We present an orner n algorithm which computes the convex hull of a two-dimensional non-self-intersecting polygon. The algorithm recovers much of the simplicity of the one presented by Sklansky (Sklansky 1972) and subsequently disproved. Unlike several algorithms which have been found since then, the modified algorithm executes a truly uniform (modeless) traversal of all the vertices of the polygon. This makes it possible to extend the algorithm to extract geometric information about the interior of the polygon.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Table of Contents

Letter on file

A-1

DTIC COPY INSPECTED 3

List of Figures

Abstract

We present an order n algorithm which computes the convex hull of a two-dimensional non-self-intersecting polygon. The algorithm recovers much of the simplicity of the one presented by Sklansky (Sklansky, 1972), and subsequently disproved. Unlike several algorithms which have been found since then, the modified algorithm executes a truly uniform (modeless) traversal of all the vertices of the polygon. This makes it possible to extend the algorithm to extract geometric information about the interior of the polygon.

1. Background

A simple algorithm for finding the convex hull of a polygon was described in (Sklansky, 1972). This algorithm was order n, and used a stack to support a backtracking technique. Subsequently, A. Bykat (Bykat, 1978) found that Sklansky's algorithm fails in some cases. Recently several algorithms have been published (Bhattacharya, 1984) (Graham, 1983) which overcome these failures at the expense of increased algorithmic complexity.

It is hoped that the algorithm presented here will prove useful because it is simpler than previous algorithms, and because unlike them it explores the interior of the polygon. It can be used as a foundation for other algorithms which extract useful geometric information about the interior of the polygon (Peshkin, 1985).

We assume that the polygon is described as a sequence of vertices in the plane. The vertices form a closed, non-intersecting chain. They are numbered 0 through n-1 for a counter-clockwise (CCW) traversal of the polygon's perimeter. Vertex n is defined for convenience as being identical to vertex 0.

We also require that vertex 0 (and n) be a point on the convex hull. This condition is satisfied by choosing vertex 0 so that it has the most negative x coordinate.

Sklansky's algorithm is simple and intuitive. It is based on a stack, which in the end contains the vertices of the convex hull. Initially the stack contains vertices 0 and 1. sp is a pointer to the top element of the stack.

for i = 2, n

while i right of ray($stack_{sp-1}$, $stack_{sp}$)
pop *discard the top element*

push i *push i to the stack*

end

The Sklansky algorithm works by considering a triplet of vertices: next-to-top-of-stack, top-of-stack, and a new vertex i. The top-of-stack vertex is rejected if the triplet forms a right turn.

The Sklansky algorithm sometimes fails, finding a left-turning sequence of vertices which self-intersects. Figure 1-1 shows the counterexample found by Bykat, and (dotted) the result of the

algorithm when applied to it. The algorithm fails because after vertex 2 is discarded, the triplet (1, 3, 4) is a left turn, and vertex 3 is not discarded.

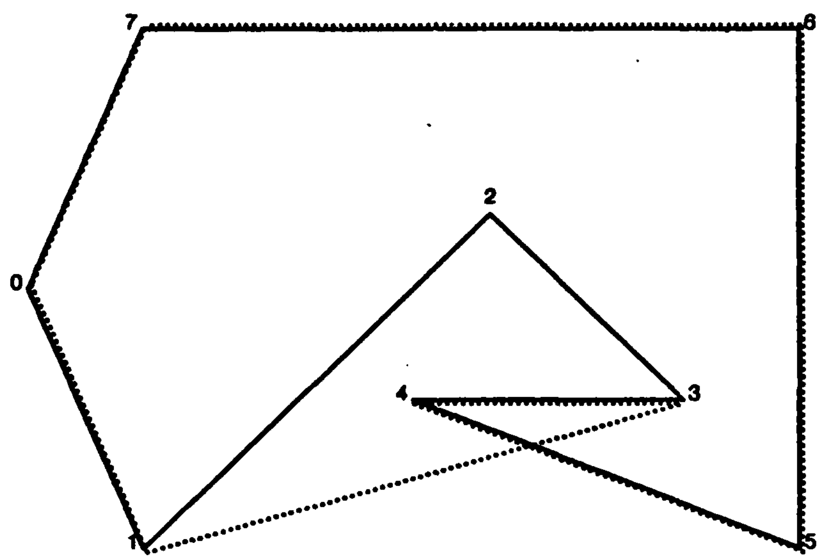


Figure 1-1: Bykat's counterexample to Sklansky's algorithm

2. The Modified Algorithm

The algorithm can be made to work properly in all cases if we can detect a class of situations like Bykat's counterexample. In Figure 1-1, for example, the situation is detected when i becomes 4. The proper response is always to reject (pop) the top-of-stack vertex, which in the example is vertex 3. The situation can be detected by comparing the angle of ray(i-1, i) as computed by two different methods, called the *cumulative angle* and the *path angle*. Disagreement of the two indicates that the top vertex on the stack should be discarded regardless of whether it forms a left turn.

The *cumulative angle* is the angle of a ray(i-1, i) computed by following the rotation of the polygon, starting at vertex 0. If the positive x axis is used to define 0 degrees, then ray(0,1) is -70 degrees, ray(1,2) is +45 degrees, ray(2,3) is -45 degrees, and ray(3,4) is -180 degrees.

The *path angle* is the angle of a ray(i-1, i) computed by following the vertices on the stack, starting at vertex 0. Ray(0,1) is -70 degrees, ray(1,3) is +15 degrees, and ray(3,4) is +180 degrees.

The cumulative and path angles always differ by a multiple of 360 degrees. Therefore it is not necessary to compute them with any precision; computation of the cumulative *quadrant* and path *quadrant* is sufficient. Since the remainder of the algorithm requires only a left/right comparison, there is no need to compute trigonometric functions at all. If trigonometric functions *are* used, some care must be exercised in testing equality of the cumulative and path angles, otherwise round-off errors may cause disagreement to be reported when in fact there is none.

In the description of the algorithm which follows, we have computed the cumulative and path angles by using trigonometric functions (implicit in the CCW function). This makes the algorithm easier to understand, and avoids the uninteresting programming details of quadrant counting. Since the path angles are then already available, we have described the left/right test in terms of path angles as well.

In the algorithm, **cumangl** is maintained as the cumulative angle of the edge (i-1, i). $\mathbf{Pathangl}_j$ is at all times maintained as the path angle of the ray ($\mathbf{stack}_{j-1}$, $\mathbf{stack}_j$).

To verify the correctness of the algorithm, we have tested it on 25000 randomly generated polygons, of which Figure 2-1 is a typical example. Results were compared with the output of an order n^2 algorithm which treats the vertices of the polygon as an unordered collection of points. The modified algorithm found the correct convex hull in each case.

A straightforward implementation of the modified algorithm in the language "C" was compared to a similar implementation of Sklansky's algorithm. On a VAX-780 the modified algorithm ran at about 1/5 the speed of Sklansky's algorithm, primarily due to the computation of arctangents. When only *quadrants* were computed, to avoid the arctangents, the algorithm ran at about 1/3 the speed of Sklansky's algorithm. The small improvement is due to the greater complexity of computing the quadrants.

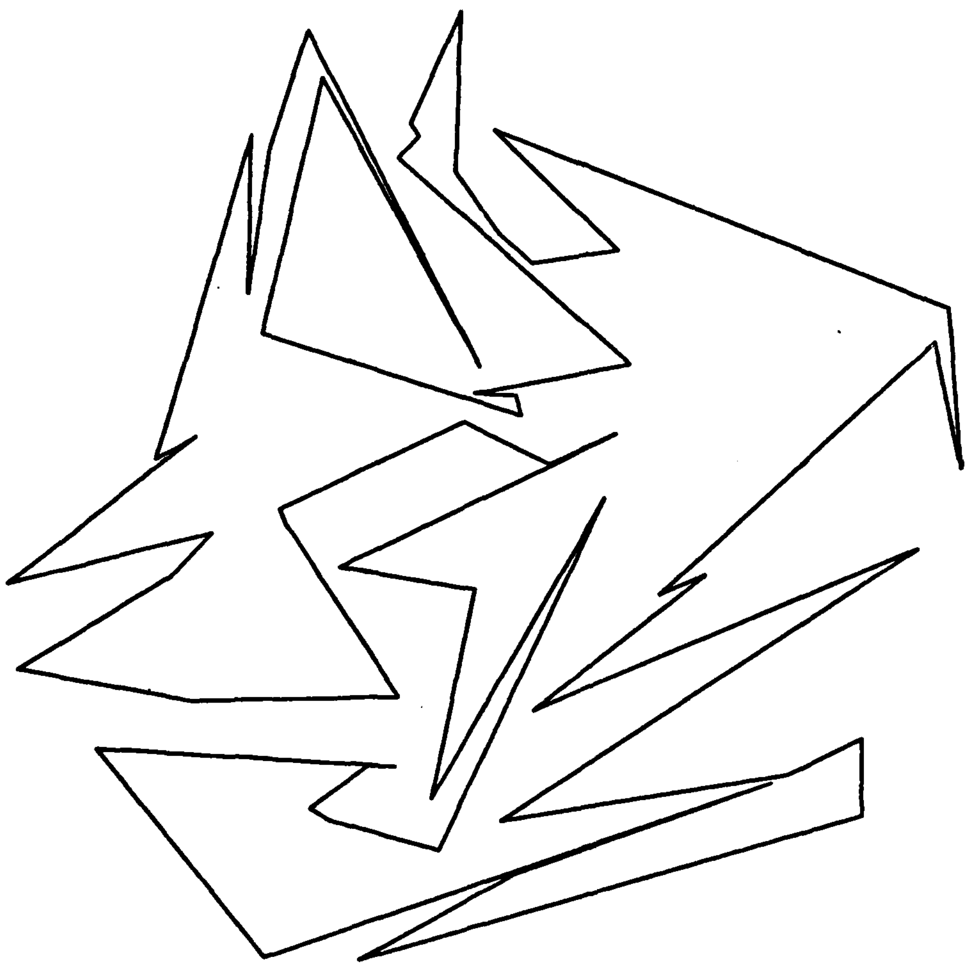


Figure 2-1: A typical polygon used for testing the algorithm

3. Convex Hull Algorithm

INPUT:

(x_i, y_i), for $0 \leq i \leq n$, are the cartesian coordinates of the vertices of the polygon.
Vertex 0 is extremal in the negative x direction.
Vertices 0 and **n** are equivalent.
The polygon is traversed in a CCW sense with increasing subscript.

OUTPUT:

$\mathbf{stack}_i$, for $0 \leq i \leq n$, are the vertex numbers of the vertices on the convex hull.

THE ALGORITHM:

definition of function **CCW(k, j, i)**, ranging from $-\pi$ to π:

if (**k** = -1) **CCW(k, j, i)** = the CCW angle from the positive x axis to ray((x_j, y_j) (x_i, y_i))

if (**k** $\geq$ 0) **CCW(k, j, i)** = the CCW angle from ray((x_k, y_k) (x_j, y_j)) to ray((x_j, y_j) (x_i, y_i))

initialize

$\text{stack}_0 \leftarrow -1$
$\text{stack}_1 \leftarrow 0 \qquad \text{stack}_2 \leftarrow 1$
$\text{pathangl}_1 \leftarrow 0 \qquad \text{pathangl}_2 \leftarrow \text{CCW}(-1, 0, 1)$
$\text{cumangl} \leftarrow \text{pathangl}_2$
$\text{sp} \leftarrow 2$

compute

```
for i = 2 to n

        cumangl ← cumangl + CCW(i-2, i-1, i)          update angles
        pathangl_{sp+1} ← pathangl_{sp} + CCW(stack_{sp-1}, stack_{sp}, i)

        if pathangl_{sp+1} - cumangl > .1             test for disagreement

                                                      reject i-1
                sp ← sp - 1
                pathangl_{sp+1} ← pathangl_{sp} + CCW(stack_{sp-1}, stack_{sp}, i)

        while sp > 1 and pathangl_{sp+1} ≤ pathangl_{sp}    test for right turn

                sp ← sp - 1                           reject stack_{sp}
                pathangl_{sp+1} ← pathangl_{sp} + CCW(stack_{sp-1}, stack_{sp}, i)

        sp ← sp + 1
        stack_{sp} ← i
end
```

4. Acknowledgements

The authors acknowledge the helpful comments of Gerard Cornuejols and Chris Van Wyk. We also thank the anonymous referee who found a mistake in a previous algorithm.

This work was supported by a grant from the Xerox Corporation, and by the Robotics Institute, Carnegie-Mellon University.

References

B. Bhattacharya and H. Elgindy. A New Linear Convex Hull Algorithm for Simple Polygons. *IEEE Transactions on Information Theory*, Jan 1984, *IT-30(1)*, 85-88.

A. Bykat. Convex Hull of a Finite Set of Points in Two Dimensions. *IPL*, Oct 1978, *7(6)*, 296-298.

M. A. Peshkin and A. C. Sanderson. *Reachable Grasps on a Polygon: The Convex Rope Algorithm*. Technical Report CMU-RI-TR-85-6, Carnegie-Mellon University Robotics Institute, 1985.

J. Sklansky. Measuring Concavity on a Rectangular Mosaic. *IEEE Transactions on Computers*, Dec 1972, *C-21(12)*, 1355-1364.

END

FILMED

11-85

DTIC